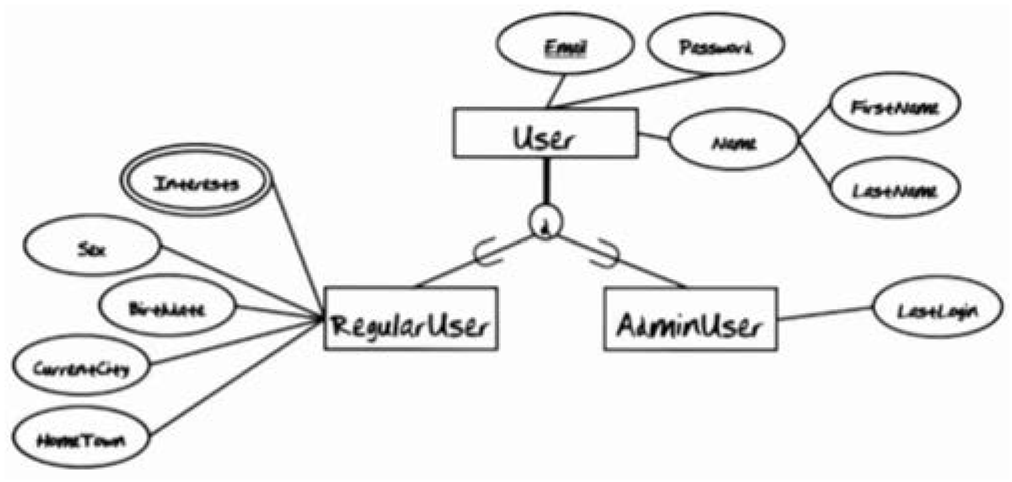
Methodology II: SPECIFICATION

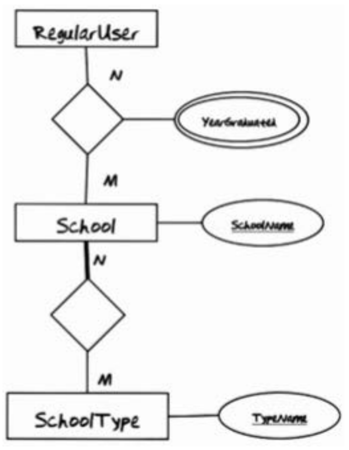
Notes for CS 6400: Database Systems Concepts & Design
Georgia Tech (Dr. Jay Summet with Dr. Leo Mark Christensen), Summer 2017
as recorded by Brent Wagenseller

Requirements (continued from Methodology I)

- <u>(BRENTS NOTE: Recall Dr. Christensen is speaking to his mock project, NOT YOUR ACTUAL PROJECT; anything specific about this particular EER or database may or may not pertain to your project)</u>
- This slide re-emphasizes a primary key (underlined) and composite attributes (name to first / last name)
- When possible, use the names that are provided in the supplied documentation as attribute names in your constructed EER diagram
    - this way, the diagram is tied closely to the real world
    - Reality is represented by the documents the people requesting the application presented to you; this needs to bear considerable weight when constructing the EER
- Additional constraints on the user



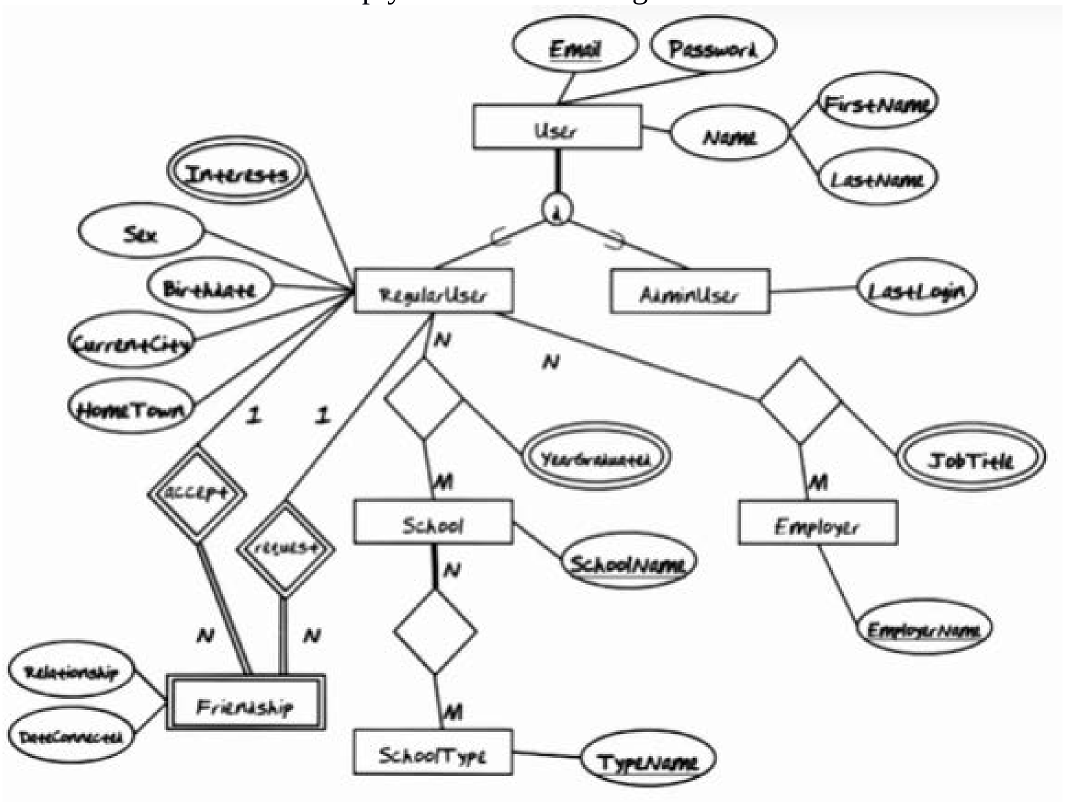
    - Its noted here that a user can be an admin OR a regular user, but cannot be both – this means any given user CANNOT have both a 'LastLogin' attribute AND a 'CurrentCity' attribute
    - Both admins and regular users have an email, login, and name (first / last)
    - Note the use of the disjointness constraint that ties User to RegularUser / AdminUser; this signifies the OR (as in, MUST be a RegularUser OR an AdminUser, NOT BOTH, NOT NEITHER)
- Handling multiple schools



- One user/school pairing must have a 'YearGraduated', even if its NULL
  - The sticking point is one student may attend undergrad, grad, and doctorate at the same school – so there must be a 3rd identifier that identifies school type (undergrad/grad/etc)
  - Technically only the 'YearGraduated' attribute is labelled as a multi-value property
    - This is similar to another requirement. User/employer/jobTitle
      - In this case, Dr. Christensen describes jobTitle as the sole multi-value because "one user can have different job titles for the same employer"

EER Diagram

- NOTE: Dr. Christensen simply went over the diagram below in this slide



Reading EER Diagrams

- Its possible to find programs that will read an EER diagram in English
- Having the EER translated to English like this would be helpful
- That said be careful – make SURE it interprets your diagram properly! Read it / check if yourself!

Data Formats

- A good way to gague the data type of the various attributes is to look at examples given in the input / output documents provided by the customer
    - <u>You may have to clarify / consult with the customer if it is not clear</u>
- Basic attributes: CHAR / VARCHAR / DATETIME / INT / FLOAT (or DOUBLE, depending)
- If the attribute looks like a character string, use a VARCHAR or CHAR
- If the attribute will be a number, use INT or FLOAT (depending)
- It usually makes sense to put bounds on attributes too (for example, password no less than 8 characters)
- Addresses can be particularly hard to deal with
- Rule of thumb: Beg, steal, borrow data formats!
    - The point here is that if someone else put in a bunch of time and effort into developing something, at least consider using what they developed

Constraints

- **Constraints** are a set of rules that we need to follow
    - Sometimes they can be enforced in the database itself
    - Sometimes it cannot be described in the database, so it must be done via programming
- Constraint examples
    - DateConnected is NULL until request is accepted
    - Cannot be a Friend with yourself
    - Can only comment on status of friends
    - Data formatting constraints
    - Constraints that can be expressed in the EER diagram

Web Apps Vs Traditional Apps

- Web Apps
    - Traditionally, almost stateless
        - This means very little variables can be / are used
    - Must have some state, e.g. email of the session user
        - These are stored in session variables in PHP
    - May need some click stream history
    - Things are changing; so-called web 2.0 and AJAX technologies provide more rich

    user interface in the web browser. These technologies make it easier to keep state information locally on the browser or behind the scenes on the server (but not in the DB)
  - In this sense, the web apps are beginning to act more like traditional apps
- Traditional Apps
  - In a traditional app, it is much easier to manage local state separately from the DB (e.g., using smart widgets, etc.)
  - A whole slew of changes can be collected before submitting them all to the database
    - In other words, a massive amount of data can be collected before anything is actually pushed to the database; this can cut down on transactions if multiple edits are needed
  - Supports better control of ACID transactions execution
- Web apps are moving closer to traditional apps

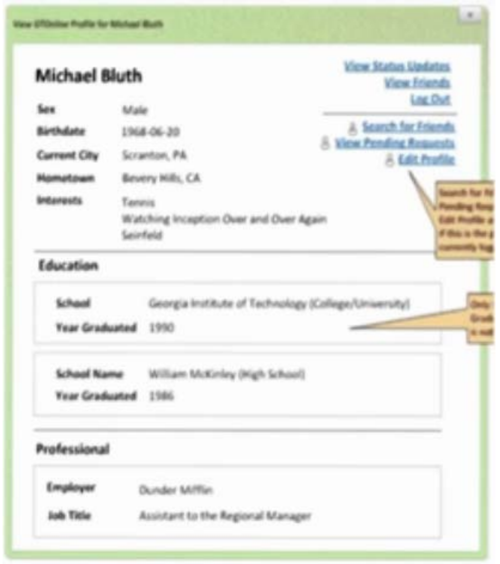
Task Decomposition

- For each one of the tasks we identified in the Information Flow Diagram (IFD), we need to figure out if it's a single task or if it needs to be decomposed
- Rules of thumb
  - <u>Lookup task –OR- is it a modification to the database (update/insert/delete)</u>
    - If many actions take place, the task may need to be decomposed
    - This is important, as the database needs to lock the table (so no other process can use it) under some circumstances when modifying the table, but lookups usually come without said locks
  - <u>How many schema constructs are involved?</u>
    - How big a portion of the database is involved in the operation
    - Not only could this enact database locks as described above, it could trigger database locks across multiple tables
      - If an action needs more locks, it will tie up more tables to perform that task
      - It will also, in theory, take longer to execute as it has to wait for the locks to be free before proceeding
    - If a large number of schema constructs are involved, you may want to decompose the task into smaller tasks using smaller portions of the database
  - <u>Are enabling conditions consistent across tasks?</u>
    - This is based on what enables different portions of a task
    - If some portions are enabled, run them, instead of waiting when the entire thing can run – the resources may not be available!
    - Smaller portions are easier to run, so break these off if possible
    - Let run what can run – scheduling
  - <u>Are frequencies consistent across tasks?</u>
    - If the tasks contain actions that are done with a high frequency and actions that have a low frequency, split them up

        - High frequency requests will need an index to speed them up
        - Low frequency requests may not need indexing
      - Index only what must be indexed
        - Indexing takes physical space to keep, so this must be a consideration
    - <u>Is consistency essential?</u>
      - (ACID transaction properties)
      - Is it essential that all the pieces of a task get done in one transaction, or is it OK if it takes multiple transactions
        - For example, transferring money from one account to another NEEDS to be done at once, otherwise the tables could become inconsistent!
    - Is mother task control needed or not?
      - For example, the example of transferring money from account to account needs a mother task

Task Decomposition: Example

- Here is the 'view profile' task:



View GTOnline Profile for Michael Bluth

Michael Bluth

Sex	Male
Birthdate	1968-06-20
Current City	Scranton, PA
Hometown	Bevery Hills, CA
Interests	Tennis Watching Inception Over and Over Again Seinfeld

Education

School	Georgia Institute of Technology (College/University)
Year Graduated	1990

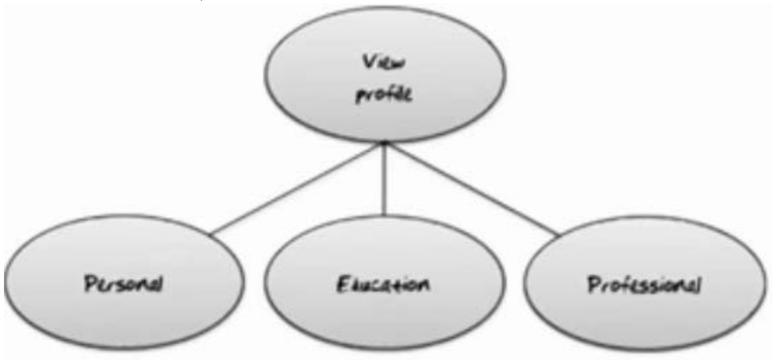
School Name	William McKinley (High School)
Year Graduated	1986

Professional

Employer	Dunder Mifflin
Job Title	Assistant to the Regional Manager

- Should we decompose this task further? Look at the facts
- View Profile Facts
  - Three lookups for regular users: personal, education, and professional information
  - All three are read-only
  - All three are enabled by a user's login or a friend's lookup
  - All three have the same frequency
  - Several different schema constructs are needed
  - Consistency is not critical, even if the profile is being edited by the user while a friend is looking at it
  - They can be done in any order

    - All three must be done, so a mother task is needed



    - <u>Most of these indicate we should decompose the task into three sub-tasks</u>

Abstract Code (Pseudocode)

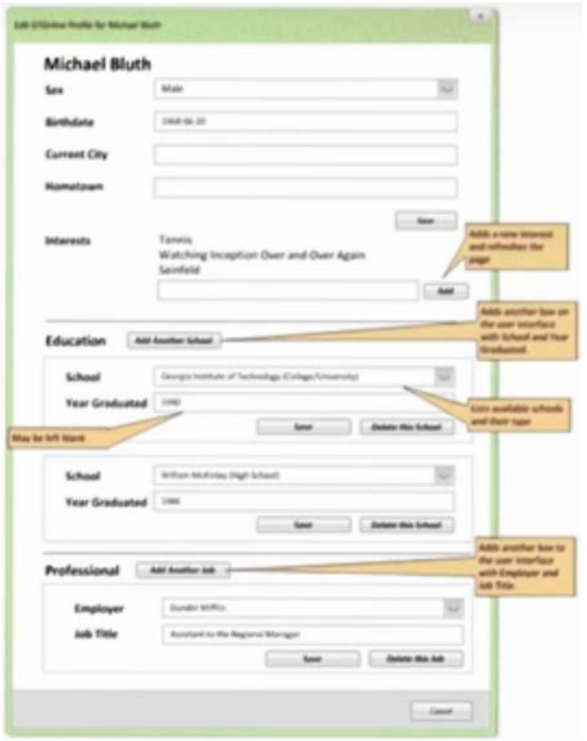
- Pseudocode uses Python-like indentation (so indents can represent if statements or loops)
- Don’t get hung up on the pseudocode displayed here; as long as it conveys your thoughts in a consistent manner and lays everything out your pseudocode is fine
- Pseudocode for ViewProfile

```
Find the current User using the User Email;
Display User Name;
Find the Current RegularUser using the User Email;
Display RegularUser Sex, Birthdate, CurrentCity, Hometown, and Interests;
Find each School for the RegularUser
        {Display School Name and YearsGraduated;
        Find School Type;
        Display SchoolType Name};
Find each Employer for the RegularUser
        Display Employer and JobTitles
```

TaskDecomposition - EditProfile

- Here is another example (picture on next page)
- EditProfile Facts:
    - Lookups of Personal, Education, and Professional information of a RegularUser (use: view profile)
        - Need to get information before we can edit it – simply re-use the view profile task already created
    - Lookups of School and Employer lists
    - Edits of Personal, Education, and Professional information
    - Read, insert, delete, update
    - All three are enabled by a user’s login and separate edit request
    - Different frequencies
    - Several different schema constructs are needed

    - Consistency is not critical, even if the profile is being looked at by a friend of the user
    - Lookup done first followed by any number of edits and lookups
    - Mother task needed
- EditProfile Page:



Edit GTOnline Profile for Michael Bluth

Michael Bluth

Sex: Male

Birthdate: 1968-06-20

Current City:

Hometown:

Interests: Tennis, Watching Inception Over and Over Again, Seinfeld

Education

School: Georgia Institute of Technology (College/University)
Year Graduated: 1990

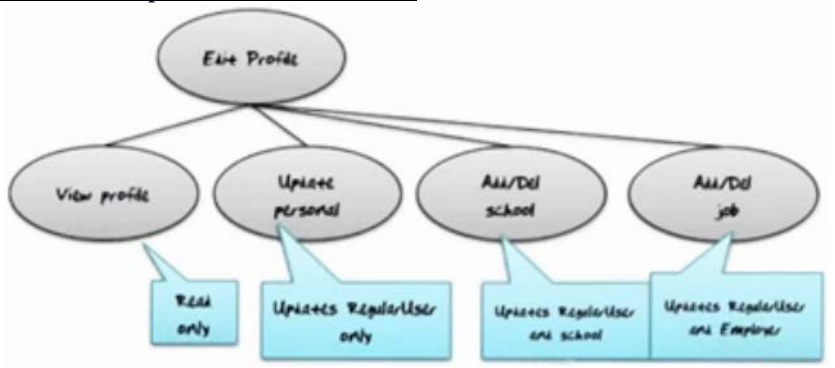
School: William McKinley (High School)
Year Graduated: 1986

Professional

Employer: Dunder Mifflin
Job Title: Assistant to the Regional Manager

Save
Add
Add Another School
Save
Delete this School
Add Another Job
Save
Delete this Job
Cancel
Adds a new interest and refreshes the page
Adds another box on the user interface with School and Year Graduated
Lists available schools and their type
May be left blank
Adds another box to the user interface with Employer and Job Title

- <u>This must be decomposed into sub-tasks:</u>



    - ‘View Profile’ is read-only (already covered)
    - ‘Update personal’ updates RegularUser only
    - ‘Add/Del school’ could update both RegularUser and school
    - ‘Add/Del job’ could update both RegularUser and employer

Abstract Code / Pseudocode for EditProfile

- <u>Pseudocode ties the input / output documents (which sits outside the system boundary) through the task edit profile to the database</u>
- Also note that the pseudocode has some formatting

    - Bold / Underlined is the task definition
    - Bold is the Task Call
    - Italics are button names
- Pseudocode for Edit Profile:
  <u>EditProfile</u>
  View Profile. Populate School and Employer dropdowns
  While no buttons are pushed, do nothing.
  While a button is pushed, do the following:
   {If SAVE PERSONAL: **Update Personal. View Profile.**
  If DELETE THIS SCHOOL: **Delete School. View Profile.**
  If DELETE THIS JOB: **Delete Job. View Profile.**

  If ADD ANOTHER SCHOOL: **View Profile;** display the form with room for another School;
  If ADD ANOTHER JOB: **View Profile;** display the form with room for another Job;

  If SAVE SCHOOL: **Add School. View Profile**
  If SAVE JOB: **Add Job. View Profile**

  If CANCEL: Go to **View Profile** for current User};

Specification Wrap-Up

- This completes the ‘Specification’ phase of the Database Development Methodology
- We have
    - Generated an Extended Entity Relationship Diagram
    - We looked at
        - Data formats
        - Constraints
    - We then looked at the task decomposition and wrote abstract code
- We now need to proceed with the design
    - We will translate the EER to a Relational Database Schema
    - We will take the abstract code and develop the supporting SQL that will interact with the relational database